European Patent Office
Postbus 5818
2280 HV RIJSWIJK
NETHERLANDS
Tel.: +31 70 340 2040
Fax: +31 70 340 3016

Europäisches Patentamt | European Patent Office | Office européen des brevets

Granleese, Rhian Jane
Marks & Clerk
90 Long Acre
London WC2E 9RA
GRANDE BRETAGNE



EPO Customer Services
Tel.: +31 (0)70 340 45 00

Date
18.07.07

| Reference | Application No./Patent No. |
|---|---|
| EPP89914 | 04251822.5 - 1243 |

Applicant/Proprietor
KABUSHIKI KAISHA TOSHIBA

## COMMUNICATION

The European Patent Office herewith transmits as an enclosure the European search report (under R. 44 or R. 45 EPC) for the above-mentioned European patent application.

If applicable, copies of the documents cited in the European search report are attached.

☑ Additional set(s) of copies of the documents cited in the European search report is (are) enclosed as well.

The following specifications given by the applicant have been approved by the Search Division :

☑ Abstract ☑ Title

☐ The abstract was modified by the Search Division and the definitive text is attached to this communication.

The following figure will be published together with the abstract : 32

### Refund of search fee

If applicable under Article 10 Rules relating to fees, a separate communication from the Receiving Section on the refund of the search fee will be sent later.

European Patent Office

# EUROPEAN SEARCH REPORT

Application Number
EP 04 25 1822

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (IPC) |
|---|---|---|---|
| X | US 6 021 479 A (STEVENS LUIS F [US])<br>1 February 2000 (2000-02-01)<br>* abstract *<br>* column 1, line 65 - column 3, line 13 *<br>* column 4, line 25 - line 67 *<br>* column 6, line 1 - column 26, line 41; claims 1-39; figures 1a-14 * | 1-12 | INV.<br>G06F12/02<br>G06F12/10 |
| X | WO 84/01043 A (WESTERN ELECTRIC CO [US])<br>15 March 1984 (1984-03-15)<br>* abstract *<br>* page 4, line 12 - page 7, line 23 *<br>* page 8, line 1 - page 21, line 30; claim 1; figures 1-4 * | 1-12 | |
| X | WO 00/36509 A (UNISYS CORP [US])<br>22 June 2000 (2000-06-22)<br>* abstract *<br>* page 2, line 20 - page 8, line 27 *<br>* page 14, line 5 - page 16, line 5 *<br>* page 18, line 7 - page 35, line 13 *<br>* page 59, line 5 - page 76, line 10 *<br>* figures 1,2 * | 1-12 | |

TECHNICAL FIELDS SEARCHED (IPC)

G06F

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| The Hague | 10 July 2007 | Wierzejewski, Piotr |

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document



EPO FORM 1503 03.82 (P04C01)

## ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 04 25 1822

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

10-07-2007

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 6021479 A | 01-02-2000 | CA 2247157 A1 | 19-03-1999 |
| | | US 6336177 B1 | 01-01-2002 |
| WO 8401043 A | 15-03-1984 | DE 3372677 D1 | 27-08-1987 |
| | | EP 0116591 A1 | 29-08-1984 |
| | | JP 59501802 T | 25-10-1984 |
| | | US 4539637 A | 03-09-1985 |
| WO 0036509 A | 22-06-2000 | AT 359550 T | 15-05-2007 |
| | | BR 9916308 A | 11-12-2001 |
| | | CA 2355065 A1 | 22-06-2000 |
| | | EP 1145122 A2 | 17-10-2001 |
| | | JP 2002532806 T | 02-10-2002 |
| | | JP 2006216068 A | 17-08-2006 |

**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 3 :
G06F 15/16, 9/46

A1

(11) International Publication Number: WO 84/ 0104

(43) International Publication Date: 15 March 1984 (15.03.84

(21) International Application Number: PCT/US83/00435

(22) International Filing Date: 28 March 1983 (28.03.83)

(31) Priority Application Number: 411,899

(32) Priority Date: 26 August 1982 (26.08.82)

(33) Priority Country: US

(71) Applicant: WESTERN ELECTRIC COMPANY, INC. [US/US]; 222 Broadway, New York, NY 10038 (US).

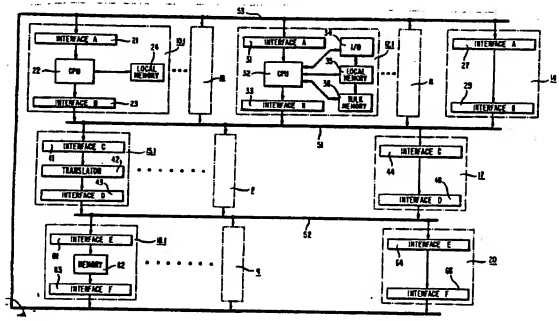
(72) Inventor: DeBRULER, Dennis, L. ; 4720 Main Street, Downers Grove, IL 60515 (US).

(74) Agents: HIRSCH, A., E., Jr. et al.; Post Office Box 901, Princeton, NJ 08540 (US).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), FR (European patent), GB (European patent), JP, LU (European patent), NL (European patent), SE (European patent).

Published
*With international search report.*

(54) Title: METHOD AND APPARATUS FOR HANDLING INTERPROCESSOR CALLS IN A MULTIPROCESSOR SYSTEM




(57) Abstract

A multiprocessor arrangement in which the individual program functions of a program process are executed on different processors (10.1). Data shared by different program functions is stored in shared memory and the programs are stored in local memory (24) of the individual processors. One processor calls for the execution of a program function by another processor by causing the program address and a point to the program function context to be loaded into a work queue of the called processor. Input output modules (34) are treated as processors. Facilities are provided for the transfer of blocks of data over the interconnection bus system. Virtual addresses are translated to physical addresses in one facility common to all processors.

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | |
|---|---|---|---|
| AT | Austria | LI | Liechtenstein |
| AU | Australia | LK | Sri Lanka |
| BE | Belgium | LU | Luxembourg |
| BR | Brazil | MC | Monaco |
| CF | Central African Republic | MG | Madagascar |
| CG | Congo | MR | Mauritania |
| CH | Switzerland | MW | Malawi |
| CM | Cameroon | NL | Netherlands |
| DE | Germany, Federal Republic of | NO | Norway |
| DK | Denmark | RO | Romania |
| FI | Finland | SE | Sweden |
| FR | France | SN | Senegal |
| GA | Gabon | SU | Soviet Union |
| GB | United Kingdom | TD | Chad |
| HU | Hungary | TG | Togo |
| JP | Japan | US | United States of America |
| KP | Democratic People's Republic of Korea | | |

# METHOD AND APPARATUS FOR HANDLING INTERPROCESSOR CALLS IN A MULTIPROCESSOR SYSTEM

## Technical Field

This invention relates to multiprocessor systems and more specifically, to means for transferring data and program control in such systems.

## Background of the Invention

A multiprocessor system is a data processing system in which a number of processors cooperate to execute the total overall task of the system. It is used when one processor cannot handle the full data processing load demanded of the system. While a great deal of progress has been made in solving the problems of multiprocessor systems having a small number of processors, no satisfactory arrangement exists for the achievement of a very high throughput in a system having a large number of modest performance processors.

Multiprocessor systems, in common with other data processing systems, use random access storage such as semiconductor random access memories, and bulk storage, such as magnetic disks or tapes. When a particular task is being performed by the system, the program and data associated with this task is stored in random access memory so that the data can be processed. At other times, the data is stored in bulk storage, ready to be transferred or paged into random access storage when the need arises.

No really satisfactory technique exists in prior art systems for efficiently and economically sharing random access memory, especially that containing programs, among many processors. Some prior art systems share all random access memory, including programs and data, among all processors. When program memory is to be fully shared

among all processors, a bottleneck exists in accessing and communicating program instructions from common memory to each of the processors upon demand. Either an extremely high throughput bus or a complex bus interconnection scheme is used to transmit the instructions from the memory to several processors. Such prior art buses are expensive and the systems are limited to a small number of processors since they require the sending of vast quantities of program instructions over the buses with an inevitable loss of performance capability.

In other prior art systems, local random access memory is provided for each processor. Multiprocessor systems generally operate in the multiprocessing mode, wherein the system executes a number of broad tasks, called program processes, simultaneously. Associated with each program process are a number of variables and parameters, stored in an area of memory called the program function context. Each of these program processes accomplishes its objectives by executing a number of sub-tasks or program functions which utilize the data of the associated program function context. In prior art multiprocessing systems, a program process is usually confined to a single processor. Placing an entire program process on one processor requires an expensive, large local memory for that processor and degrades performance, by requiring a great deal of manipulation of memory contents. The alternative of breaking large processes down into small processes is also inefficient and leads to an unwieldy software structure.

Prior art multiprocessor systems use restricted and specialized communication means between processors and input/output controllers and among input/output controllers in order to avoid overloading the common system bus. Input/output controllers are associated with various combinations of devices such as magnetic disk or tape memories, input/output terminals and displays, high speed printers, punched card readers and punchers.

Usually, these controllers are interconnected by arrangements with limited access; all processors cannot directly access all input/output units without considerable expense. This means that system performance is degraded if large amounts of data must be exchanged between two input/output controllers which were initially designed to exchange very little data.

Many modern processors and multiprocessor systems use a highly flexible method of addressing memory called virtual addressing. A virtual address is an address of main (random access) memory in a simulated processor system; the virtual address is translated into a physical address in the actual processor system before it is used to access random access memory. The translation mechanism is flexible so that at different times, a given virtual address may correspond to different physical addresses of random access memory; a virtual address may also correspond to an address of bulk storage. Virtual addresses tend to be fixed in a program; physical addresses are assigned to a given segment of virtual addresses when needed. A page fault occurs when a virtual address does not correspond to a physical address of random access memory, i.e., when the translation mechanism fails to find such a correspondence. Page faults always require the adjustment of the translation mechanism, and sometimes, the paging of data from bulk storage into newly assigned random access memory space.

The design of economical address translation mechanisms for translating virtual addresses to physical addresses presents a problem in a multiprocessor system. In prior art multiprocessor systems, these mechanisms are implemented using very fast circuits because the delay of address translation is added to each access of random access memory. The size of address translation mechanisms is usually restricted by cost because of the high speed requirement. A result is that many page faults, i.e., system indications that a desired memory location cannot

be accessed, occur because, although the required segment is available in storage, the translation to reach that location is not currently in the address translation mechanism. In prior art multiprocessor systems with many individual processors and address translator mechanisms, address translation is expensive and tends to limit system performance. Furthermore, prior art bus schemes interconnecting the processors and shared memories of a multiprocessing system are frequently a limitation on the total throughput.

Summary of the Invention

In accordance with this invention, each processor of a multiprocessor system is adapted to cause a program function to be executed by any processor of the system, by linking a request for such execution to a work queue of the called processor. The calling processor links the program function context, specifying the called program function, to the work queue of the called processor. The called processor then executes that program function. In one embodiment of this invention, the link to the program function context is augmented by a link to a program address of the called program function; the latter link need not be provided by the program function context in this case. Alternatively, the link can be part of the data provided by the program function context.

In one embodiment of this invention, each processor has local memory directly accessible only to the central processing unit of that processor used for storing programs to be executed by the associated processor. Preferrably, shared memory means also exist, accessible to all processors, and used to link the called program function context to the work queue of another processor. Advantageously, by permitting individual program functions to be assigned to any processor, thus allowing a process to be spread over several processors, the size of local memory for each processor can be reduced. Repeated copies

of a function used by many processes can be eliminated.

In one embodiment of this invention, external bus means interconnect processors and shared memory means. Further, input/output controllers have the same access to the bus means as processors, can call or be called for the execution of a program function using the same techniques, and can similarly access shared data. Advantageously, this provides full access among all input/output controllers, processors, and shared memory.

In one embodiment of this invention, the shared memories, processors, and bus means are adapted to transfer blocks of data rapidly and efficiently by treating the words of the block together instead of as separate and independent entities. Consecutive addresses need not be transmitted repetitively for such a block transfer. Advantageously, this reduces the number of address translations required and permits the system to operate efficiently even with a slower address translation mechanism. Advantageously, the bus means are used for transferring blocks of data among input/output controllers, shared memory means, and processors, thus providing highly flexible interconnection apparatus among these units.

In one embodiment of this invention, virtual addressing is used to access memory. The virtual address translation mechanism used in one specific embodiment is a single common facility accessible by all processors, and comprises one or several independent translation modules. These modules operate on different portions of the virtual address spectrum. Advantageously, the use of a common facility makes it economically feasible to make this facility large enough to eliminate those page faults which in prior systems result from required translations not being available in the translation mechanism. In one embodiment of this invention, each program process occupies a different portion of the virtual address spectrum. Advantageously, this facilitates a relatively

uniform distribution of translation load among the translation modules. In one embodiment, data indicating the identity of a processor designated to execute a called program function and an indication of the address of that called program function is stored in memory. Advantageously, such memory can be shared memory. If virtual addressing is used in a system, such memory can be addressed using virtual addressing means. Alternatively, such data may be stored in the virtual address translation mechanism.

In an alternative embodiment of a virtual address translator, such a translator is implemented using a group of at least two serially connected blocks to perform the translation. Each block comprises memory, register and adder means. The input to the first block represents an initial virtual address, the output of each block is the input to the next block and the output of the last block includes the physical address corresponding to the initial virtual address. The blocks can be operated in a pipeline mode to allow action on several translations to proceed simultaneously. In a specific alternative embodiment, three blocks are used in tandem. The alternative embodiment of an address translator is used in an address translation module of which there may be one or a plurality of independent modules.

In one embodiment of this invention, processors are arranged to generate physical addresses after having generated a translation from a virtual to a physical address. The address translation means are adapted to transmit such physical addresses without translation. Physical addresses may, for example, occupy a dedicated portion of the total address spectrum. This arrangement reduces the number of translations required and permits the system to operate efficiently even with slower address translation mechanisms. Advantageously, the use of slower address translation mechanisms makes it economically possible to greatly expand the address capacity of these

mechanisms, thus reducing page faults.

In one embodiment of this invention, the bus means are split into three segments operating simultaneously and independently. One segment transmits signals from processor output means to virtual address translation means. A second segment transmits signals from the output of virtual address translation means to shared memory addressing and data input means. A third segment transmits signals from shared memory output means to processor input means. Intersegment connections are provided for cases in which the output of one bus segment can be transmitted directly to the next bus segment. In addition, each segment of the bus system can carry many simultaneous transactions. Each memory result and each data word to be written is tagged with the identification of the requesting or sending unit respectively so that several memory responses and several data words to be written can be interleaved. The bus means can also be used for transmitting data among processors and input/output units. Advantageously, a bus system in accordance with this invention provides high throughput without the prohibitive expense of prior art high performance buses.

Brief Description of the Drawing

The invention will be better understood from the following detailed description when read with reference to the drawing in which:

FIG. 1 is a block diagram of a multiprocessor system representing an illustrative embodiment of the invention;

FIG. 2 is a memory layout of memory entries used in executing programs in the system of FIG. 1; and

FIG. 3 is a memory layout of work queues for the processors of the system of FIG. 1.

Detailed Description

FIG. 1 is a block diagram of an illustrative multiprocessing system consisting of processors 10.1,...,m, input/output (I/O) modules 12.1,...,n, address translators 15.1,...,p and shared memory modules 18.1,...,q, interconnected by bus system 51, 52, 53. The processors are identical in structure and each includes a central processing unit (CPU) 22, a local memory 24, and bus interface circuits 21 and 23. The CPU 22, which may be a commercially available unit such as the Motorola MC 68000, is connected to the local memory 24 which stores programs and data dedicated to the CPU. The interface circuits 21 and 23 provide an interface between the CPU and buses 51 and 53. The I/O modules 12.1,...,n are identical in structure and each comprises a central processing unit (CPU) 32, interface circuits 31 and 33 by which the CPU is connected to buses 51 and 53, and input/output equipment 34, local memory 35, and bulk memory 36 all connected to the CPU. The CPU 32 may be the same kind of machine as the CPU 22. Local memory 35 is adapted to store data and the programs for the CPU 32. The input/output equipment 34 includes input and output terminals. Bulk memory 36 consists of bulk storage devices such as magnetic disks and tapes.

Memory modules 18.1,...,q are identical in structure, each including a standard random access memory unit such as memory 62 and bus interface circuits 61 and 63. Any of the memory modules may be addressed by any of the processors or I/O modules. The memory addresses transmitted on bus 51 may be either virtual addresses or addresses of physical memory locations (physical addresses). A virtual address is the identification of an item (program or data) which may be stored at any physical address in one of the memory modules or in bulk memory in one of the I/O modules. Virtual addresses occurring on bus 51 are translated by means of the address translators 15.1,...,p into physical addresses defining

physical location in the memory modules. Each of the address translators includes a translator unit 42 and interface circuits 41 and 43. The translator units are well-known virtual address translators which include information defining the present physical location of the item identified by the virtual address.

If the data contained in the translator 42 indicates that the physical address corresponding to the translated virtual address is in one of the memory modules 18.1,...,q, the translator module will transmit the physical address to the appropriate memory module via bus 52. In the event that the corresponding physical address does not fall within the range of addresses of the memory modules, the item identified by the virtual address is obtained from the bulk memory location corresponding to the virtual address and placed in a selected one of the memory modules. Furthermore, the information in the translators is changed to reflect the selected memory module location for the virtual address. The memory access operation may be completed using the new physical address. Address tables stored in memory define the present location of all data, whether in bulk storage or in a memory module.

In this embodiment of the invention, the bus system is broken into three parts. Bus 51 is used to transmit address and data outputs of the processors and I/O modules to the translators 15.1,...,p. Bus 52 is used to transmit addresses and data to the memory modules 18.1,...,q. Bus 53 is used to transmit data from the memory modules to processors 10.1,...,m and I/O modules 12.1,...,n. Also connected between buses 51 and 52 is a pass-through unit 17 consisting of interface circuits 44 and 46, to allow signals, such as physical address signals generated by a processor or I/O module to be passed directly from bus 51 to 52. A similar arrangement allows the direct passage of signals from bus 52 to bus 53 via bypass unit 20 and from bus 53 to bus 51 via bypass

unit 14.

Each of the units connected between bus 51 and 52 have an input interface circuit C and an output interface circuit D. Similarly, each of the units connected between bus 52 and bus 53 have an input interface circuit E and an output interface circuit F, and units connected between buses 53 and 51 have an input interface circuit A and an output interface circuit B. In each instance, interface circuits identified with the same letter are identical and each of the circuits is designed to assure compatibility between the bus circuitry and the circuitry to which it is connected.

The address spectrum of the system is broken down into three parts. The total range is from 0 to $(2^{32} - 1)$. The first part, 0 to $(2^{24} - 1)$ is reserved for local addresses within each processor. The same addresses can be used by all the processors and input/output modules for addressing their own local memories. The second range, $2^{24}$ to $(2^{26} - 1)$ is dedicated to physical addresses of shared memory. Each such physical address defines a unique location in the physical memory accessed by bus 52. The third range, $2^{26}$ to $(2^{31} - 1)$ is used for virtual addresses of shared memory.

Each processor is adapted to recognize local addresses and retain these internally. Thus, local (first range) addresses are not transmitted on bus 51. The bypass module 17 is adapted to recognize addresses in the second range and to transmit such addresses directly to bus 52 without translation. Only third range addresses are translated by modules 15.1,...,p. Module 17 is also adapted to recognize and transmit data (as opposed to addresses) directly to bus 52.

In the illustrative multiprocessing system, different portions of the total virtual address spectrum are devoted to different processes. This helps to prevent unwanted inter-process interference, and makes it possible to protect memory, common to several processes but

associated at one instant with only one process, from being illegally accessed by an unauthorized process. The translation mechanism generates memory protection codes for all memory accesses, and can generate a different memory protection code for the same location if it is associated with a different process.

The translation mechanism is broken up into different modules, each of which treats a different portion of the total address spectrum. This makes it unnecessary to store any address translation in more than one module. Each process has its own virtual address set, and each address translation module then operates upon a different set of processes. This arrangement is used to equalize the load on the different address translation modules.

The use of a single overall address translation mechanism which stores each translation only once makes it economically feasible, using conventional address translation methods, to store enough translations so that page faults due to missing translation data in the translation mechanism can be substantially eliminated. An alternative is to use a translation mechanism implemented through the use of random access memory and successive look-up operations based on the process number, segment and page; since this embodiment of the invention has sharply reduced the number of translations required, such an approach is feasible and offers an economical, very large translator module.

If a processor or input/output module must perform a number of memory operations in a particular block, it can access the basic address translation tables of the system, stored here in shared memory, to generate the physical address of that block. Thereafter, the processor or input/output module can execute programs using the physical address of that block. These physical addresses (second range) are then transmitted from bus 51 to 52 by module 17, without requiring address translation.

This reduces the address translation load on the system. Alternatively, it is also possible to add a special read command to the system which would return to a requesting processor a physical address instead of the data stored at that address; such a facility would speed up the process of generating a physical address.

In a multiprocessor system, it is frequently necessary to transfer substantial amounts of data between bulk memories or from bulk memory to random access memory and vice versa. In this embodiment of the invention, this is accomplished by using buses 51, 52, and 53 to implement block transfers of data among input/output modules, the local memories of the processors, and the shared memories. The write transfer of a block of data is accomplished by sending to the appropriate memory an initial address, a block write command, and the length of a block of data, and thereafter transmitting only data. The length and initial address are stored in interface E (61) which subsequently controls the writing of the individual words of data until the block write has been accomplished. A block read is accomplished the same way except that a block read command is sent and the individual words of data are sent from the shared memory to the input/output or processor modules. Interface A (21, 31) is used to store the initial address and length of block for a block read, in order to control the action of writing into the local memory (24, 25).

In order to implement a block transfer from a processor or input/output module to another such unit, a special command to alert the destination unit of a block transfer is provided. This command includes the identification of the destination unit and is recognized by interface A (21, 31) of that unit. Interface A then interrupts the associated CPU (22, 32). The destination unit is now ready to receive the initial address and length of block for a block read or write and to accept the data words for a block write or to send out the data

words associated with a block read.

Interface circuits A through F associated with the various modules of this system are adapted to transmit signals to and receive signals from the bus system. In some cases, they are further adapted to store extra signals and to control block operations. These interface circuits are made up of registers, counters, range recognition circuits, module identification recognition circuits, first-in, first-out register stacks, bus access arbiter circuits to resolve bus access conflicts, bus receivers, and bus transmitters, all well known in the art.

Interfaces A (21, 27, 31) and B (23, 29, 33) are adapted to implement the block transfer operations, and to recognize those signals on bus 53 which are destined for a particular processor. They include counters and addressing registers to allow the successive reads or writes associated with a block transfer to be read from or stored into the correct portion of local memory. In addition, interface B is adapted to resolve bus access conflicts between different processors.

Interface C (41) in address translation modules 15.1,....p, is adapted to recognize addresses within the assigned range of the associated translator. Interface C has storage to accept a number of almost simultaneous requests for translations by the same translator module. Interface C (44) in module 17 is similarly adapted to recognize the range of addresses not requiring translation and to recognize data to be passed directly to bus 52. Interface D (43, 46) is adapted to resolve bus access conflicts to bus 52 between different address translator modules and the bypass module 17.

Interface E (61) is adapted to recognize data signals and addresses on bus 52 destined for a particular shared memory module, and is adapted to control block transfer operations. It may be desirable for some applications to further adapt interface E to accept

isolated read or write requests in the middle of a block transfer operation. In addition, interface E is adapted to accept additional requests while memory 62 is reading a given location. Interface F(63) is adapted to resolve bus access conflicts to bus 53.

Since a number of shared memory read and write transactions, including some block transactions, are taking place simultaneously, interfaces B are adapted to tag address, block length, and data signals with the identification of the requesting processor module 10.1,...,m or input/output module 12.1,...,n, and interfaces F are adapted to tag data returned from a shared memory module 18.1,...,q with the identification of the requesting module. Interfaces A are adapted to check for this tag on any response from a read or block read command. Interfaces E are adapted to recognize the tag of a previously sent address that is attached to a subsequent block length or write data signal, and to be responsive to such tagged signals. In the case of a block transfer from a processor or input/output unit to another such unit interface A of the destination unit looks for the tag of the source unit. In addition, interfaces F are adapted to generate an acknowledgement including the identification tag of the requesting processor or I/O module on a write or block write command, and interfaces A are adapted to recognize such acknowledgements. Interfaces C and D are adapted to transmit the identity of the requesting processor or input/output module along with address, block length, or data sent to shared memories.

FIG. 2 shows an alternate implementation for the translators 42 of the address translation modules. Address translator 42 comprises three random access memories (RAM's) 310, 311, and 312 each with an associated adder (320, 321, 322, respectively) for combining input signals to produce the appropriate output and an associated register (330, 331, 332, respectively). A virtual address is composed of a process number, segment

number, page number, and page offset. The process number is used to address RAM 310, while the segment number, page number and page offset are stored in register 330. The output of RAM 310 (segment address table location) is added to the segment number selected from register 330 and used as an address of RAM 311. The contents of RAM 311 (page address table location) are then added to the page number in register 331 to locate the data specifying the specific page address in RAM 312. The output of RAM 312 (page address) is added to the page offset to generate the physical address corresponding to the original virtual address. In effect, RAM 310 stores the locations of virtual address segment tables of each process in RAM 311; the increment of the segment number then locates the data in RAM 311 specifying the location of the page address table in RAM 312; the increment of the page number then locates the page address stored in RAM 312. Note that address translator 42 can work in a pipeline mode, in which each of the three RAM's is simultaneously reading data for the translation of a different virtual address. Note further that address translator 42 is composed of three essentially identical blocks 300, 301, 302 differing only in the portion of the register 330, 331, 332 connected to adder 320, 321, 322. In some systems, it is possible to use concatenation arrangements or other specialized adding arrangements for some or all of the adders 320, 321, 322. In more or less complex virtual addressing arrangements, it is possible to use only two blocks, or more than three blocks, for implementing the translator.

FIG. 3 and 4 show memory layouts associated with the flow of program control in the multiprocessor system. The individual major tasks which a data processing system must carry out in order to accomplish its total objective are usually called program processes. These processes are carried out by executing a number of subtasks or program functions. In a multiprocessor system, several processes

are normally being executed simultaneously on different processors, and using different portions of shared memory. In the multiprocessor system of this invention, the various program functions associated with a given program process need not be executed on one processor; instead, one processor can call for the execution by another processor of a program function within the same process.

Each process that is active in the system has a process control block (FIG. 3A) of memory dedicated to keeping track of status information and parameters needed by the process. The time that the process was originally invoked is stored in location 211 and the time that the most recent function being executed on this process was invoked is stored in location 212 in the typical process control block 210. FIG. 3B shows the layout of a program function context 220, which is maintained during a program process. The context is a last in, first out memory stack. The most recently generated parameters in locations 222 and variables in locations 223 can then be found by going a small number of levels into the stack. The program status indication of the program function execution is kept with the program function context in location 221.

FIG. 3C shows two process tables 230 and 240, one for each of two processes. Each table contains a list of processor identifications and addresses for all the functions used by a process. For example, as indicated in locations 234 and 231 respectively, the first function, G(1), used by the first process shown is designated to be executed on processor P(G(1)); the address inside that processor in which this first function is stored is A(G(1)). In similar fashion, as indicated in locations 235,...,236 the processors, P(G(2)),...,P(G(N)) are designated to execute functions G(2),...,G(N); as indicated in locations 232,...,233, these functions are stored at internal addresses A(G(2)),...,A(G(N)), within these processors. Similarly, the processor

identifications and addresses for the M functions H(1),H(2),...,H(M), of a second process are shown in table 240 of FIG. 3C. Alternatively, it is possible to store indicators, such as an index or the address of a pointer, from which a processor executing a program can generate a program address and/or a processor identification.

Table 230 is initialized when the system first recognizes the need to execute the first program process and loads the various program functions required to execute that process in various processors. The table entries are filled in as each program function is loaded into its processor. When virtual memory addressing is used, as in the system being described, this table is stored at a virtual address, either defined within the code of the various program functions which may call any of the program functions of this program process, or found via pointers located at such a virtual address. In a similar manner, table 240 is initialized when the system recognizes the need to execute the second program process.

Alternatively, the contents of table 230, 240, and other similar tables can be used as the primary translation table mapping from virtual to physical addresses. This primary translation table could be stored in memory and inserted into the translation mechanism as needed, or, in case the translation mechanism were implemented through the use of random access memory and successive look-up operations, could be stored directly in the address translation mechanism.

FIG. 4 shows the memory layouts of work queues which are used to allow one program function being executed on one processor to call for a second program function to be executed either on the same processor or on any other processor in the system. Block 100 shows memory associated with a first processor, block 101 with a second processor,..., block r with a last processor. In this discussion, input/output controllers are included among

processors. In a system with i processors and j input/output modules, a total of i + j such blocks would be maintained.

Each processor has associated with it a work queue such as 110 containing a list of program functions which that processor has been called upon to execute and which it has not yet executed. In the described system, there is both a regular work queue (110, 160) and a high priority work queue (120, 170), in order to allow special high priority functions to be executed before regular functions. It is possible, using techniques well known in the art, to implement more complex priority schemes in which there can be more than two priorities of tasks. Moreover, methods of interrupting the current execution of a function in order to execute a special high priority task are also well-known in the art.

In this system, in order to simplify the problems which are associated with assuring that access to the work queue of one processor is not simultaneously sought by several other processors, each processor has an associated output queue (130, 140). The calling processor loads a request for the execution of a called function into its own output queue. A separate process, controlled, for example, by a third processor, is used to examine called program function requests in the output queues of each processor and load corresponding entries into the work queue of the processor designated to execute the called program function.

Each entry in a queue includes a program address or an address in a process table, such as 230, and the address of the program function context. Each queue has a load and an unload pointer for example, 111, 112. The load pointer is controlled by the processor loading the queue, the unload pointer by the processor unloading the queue. Thus, the first processor controls load pointer 131 and writes entries such as 133 and 134 into its output queue as it executes program functions which

call for the execution of other program functions. The first processor also controls unload pointer 112 and 122 of its work queue and reads entries such as 113, 114, 123, and 124 preparatory to executing the corresponding called functions which have been requested through the execution of corresponding calling functions.

The third processor, executing a separate process, executes a program function which examines the contents of output queues such as 130 and 180 and loads entries into the work queues such as 110, 120, 160 and 170. The third processor reads entries such as 133 in the output queue of each processor. After it reads one of these entries, it changes unload pointer 132. It examines the corresponding table entry in FIG. 3C to find the selected processor and program address and writes the program address into the appropriate work queue of the selected processor. The third processor then modifies the load pointer 111 or 121 of the appropriate work queue of the selected processor. In this arrangement, there is no restriction preventing the calling and called processor from being the same. The procedure for the call is identical.

Consider now a case in which the first processor is executing a first function and recognizes that there is a need to call for the execution of a second function. Both functions are in the first process in FIG. 3C and labeled G(1) and G(2). Processors P(G(1)) and P(G(2)) are the first and second processors in this example. During the course of execution of function G(1), the first processor will have entered data into a program function context including the status of the program and the parameters and variables that function G(2) will need in order to execute. In order to execute its part of the work of the function call, the first processor will load into its output queue 130 the address of entry 235 of table 230 of FIG. 3C. The first processor will then modify the load pointer 131 to point to this new entry.

The third processor will subsequently examine the load pointer 131 and unload pointer 132 of the output queue of the first processor and will recognize that the two do not match. This is an indication that the first processor has made an entry in its output queue. The third processor will examine the word pointed to by the entry in 133 of the output queue, where it will find the identity of the processor P(G(2)) (the second processor, in this case) which will execute function G(2), and the address A(G(2)) in that processor. The third processor will use the identity of the processor P(G(2)) to find the regular work queue 160 of the second processor and will enter the address of G(2), A(G(2)), in that work queue. In addition, it will copy into work queue 160 the pointer 135 to the program function context that the second processor will need in order to execute the second function. The third processor will then increment the load pointer of the regular work queue 160 of the second processor.

Subsequently, the second processor when it has finished other work will check to see if there is additional work by looking at the load pointer and unload pointer of its work queues 160 and 170. If these do not match, for either work queue, new work has been loaded into one of its work queues. The second processor will prepare to execute the next called function whose address and the address of whose program function context it will find in its work queue. It will then update the unload pointer so that after finishing the execution of this program function, it will be prepared to execute the next requested program function.

In the case of a function call in which the calling program function expects a return, the return from the called program function is implemented as a call to the original calling function. The return address data is stored in the program function context by the original calling program function and found there by the called

program function; values derived by the called program function are stored in locations specified by the calling program function.

In this embodiment, program function context, output queues, work queues, and their associated pointers are all in shared memory. Since facilities, such as those used for block transfers, exist to allow the reading from and storage into local memory of other processors, it is possible to place some of these items in local storage. An alternate solution to the process of calling for the execution of a process, is for the calling processor to write directly into the work queue of the called processor. This by-passes the process carried out in the above example by the third processor. It is also possible to implement the work queue using a first-in, first-out register stack in interface A (e.g., 21, 31) between a central processing unit and bus 53. This would allow one processor to write a call request directly into memory associated with a called processor.

In this embodiment, the address of the program function context and of the called program function are both stored in the work queue of the called processor. An alternative solution is to store only the address of the program function context, and to store the address of the program function, or a pointer to such an address, in the program function context. Furthermore, in this embodiment, program function addresses are recorded directly in the work queue. An alternative solution is to record a pointer or other indicator of such an address.

It is to be understood that the above-described embodiment is merely illustrative of the principles of this invention; other arrangements may be devised by those skilled in the art without departing from the spirit and scope of the invention.

CLAIMS

1. A multiprocessor system for executing a plurality of program processes, each process having an associated program function context (220), each processor (10.1) adapted to enter context data into any of the program function contexts:

CHARACTERIZED IN THAT

the system further comprises a work queue means (110) associated with each of the processors and storage means (230, 240) for storing an indication of the indentity of a designated processor (234) and an indication of the address of a first program function (235);

the system is adapted to link a predetermined program function context and the indication of the address of the first program function to the work queue means associated with the designated processor; and

the designated processor is further adapted to execute the first program function using the indication of the address linked to the work queue means associated with the designated processor and the predetermined program function context.

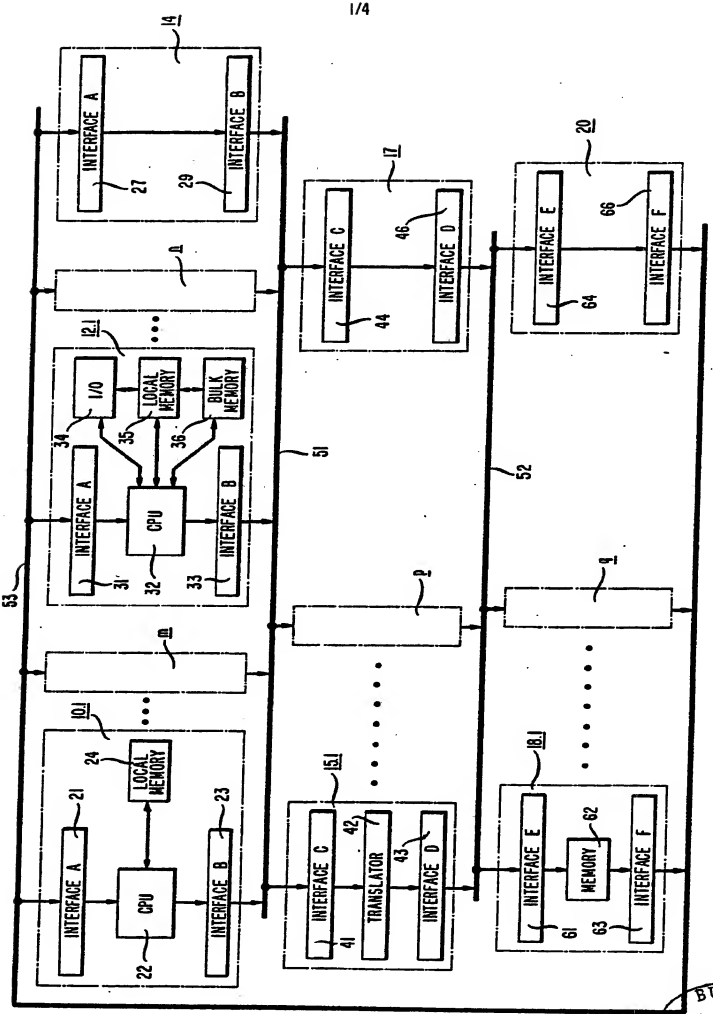



FIG. 1

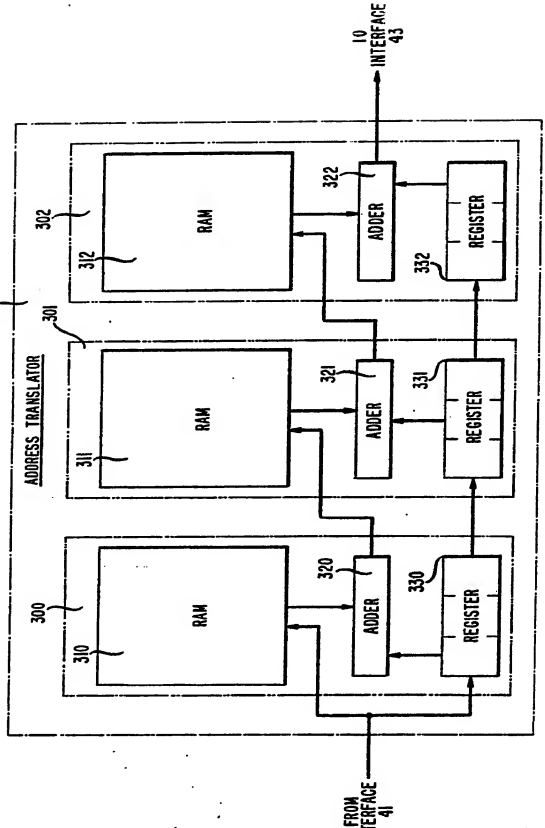



FIG. 2

FIG. 3A




FIG. 3B




FIG. 3C

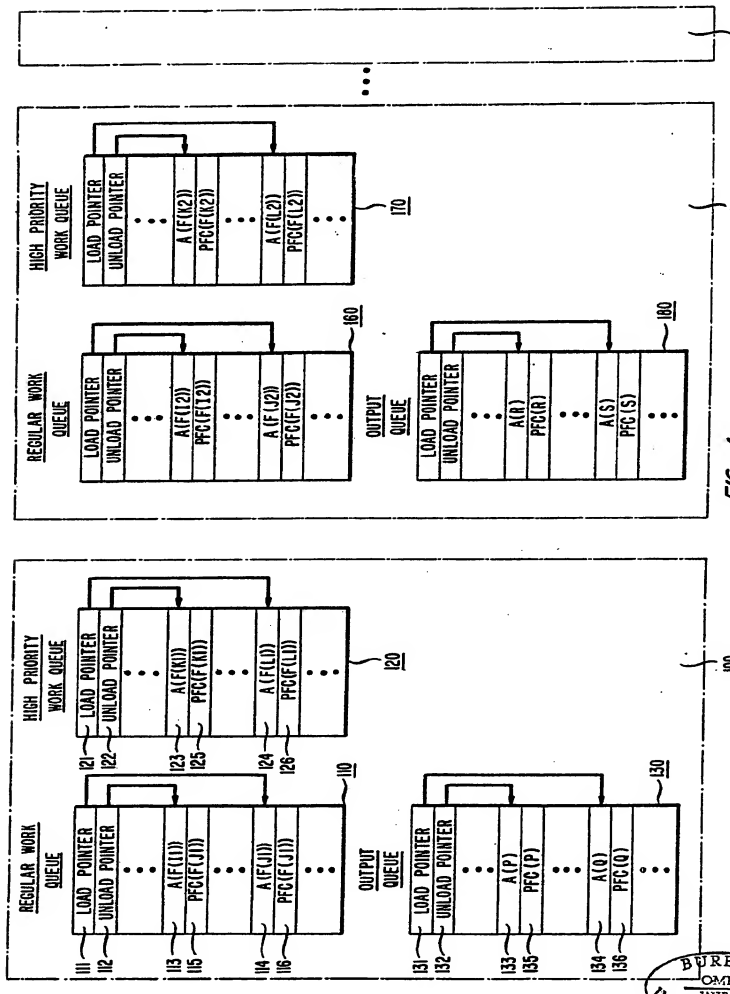



FIG. 4

# INTERNATIONAL SEARCH REPORT

International Application No PCT/US 83/00435

## I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [3]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC[3]: G 06 F 15/16; G 06 F 9/46

## II. FIELDS SEARCHED

| Classification System | Minimum Documentation Searched [4] Classification Symbols |
|---|---|
| IPC[3] | G 06 F 15/16; G 06 F 9/46; G 06 F 13/00 |

Documentation Searched other than Minimum Documentation to the Extent that such Documents are included in the Fields Searched [5]

## III. DOCUMENTS CONSIDERED TO BE RELEVANT [14]

| Category * | Citation of Document, [16] with indication, where appropriate, of the relevant passages [17] | Relevant to Claim No. [18] |
|---|---|---|
| Y | IBM Technical Disclosure Bulletin, vol. 22, no. 5, October 1979 (New York, US) Lake et al.: "Loosely coupled multi-process model applied to a line switching architecture", pages 1918-1923, see page 1918, lines 1,2, 10-20; the figure; page 1919, lines 1,2,11-16; page 1921, lines 1-4, 12-22 | 1 |
| Y | US, A, 3643227 (SMITH et al.) 15 February 1972 see column 1, lines 11-13; column 2, lines 18-31, 60-66; column 3, lines 3-11; column 5, lines 18-25; column 6, lines 24-35, 56-75; column 7, lines 1-13; column 13, lines 35-46; column 23, lines 45-55; column 30, lines 1-22; column 36, lines 44-59, 69-75; figures 1,13 | 1 |
| Y | IBM Technical Disclosure Bulletin, vol. 24, no. 2, July 1981 (New York, US) | ./. |

* Special categories of cited documents: [15]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

## IV. CERTIFICATION

| Date of the Actual Completion of the International Search [2] | Date of Mailing of this International Search Report [2] |
|---|---|
| 13th July 1983 | 10 AOUT 1983 |
| International Searching Authority [1] | Signature of Authorized Officer [20] |
| EUROPEAN PATENT OFFICE | G.L.M. Kruydenberg |

Form PCT/ISA/210 (second sheet) (October 1981)

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

| Category * | Citation of Document, 16 with indication, where appropriate, of the relevant passages 17 | Relevant to Claim No 18 |
|---|---|---|
| | Lorie et al.: "Multiprocessor system for executing concurrent sequential processes with shared variables", pages 1019-1020, see page 1019; page 1020, lines 1-16 | 1 |
| Y | GB, A, 1170587 (GENERAL ELECTRIC) 12 November 1969 see page 2, lines 5-51; the figure | 1 |
| A | American Federation of Information Processing Society Proceedings of the National Computer Conference; Dallas, 13-16 June 1977, A.F.I.P.S. Joint Computer Conference. 1977, vol. 46 (Montvale, US) Swan et al.: "The implementation of the Cm multi-microprocessor", pages 645-655 | 1 |

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON

INTERNATIONAL APPLICATION NO. PCT/US 83/00435 (SA 5041)

This Annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report. The members are as contained in the European Patent Office EDP file on 02/08/83

The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

| Patent document cited in search report | Publication date | Patent family member(s) | | Publicatic date |
|---|---|---|---|---|
| US-A- 3643227 | 15/02/72 | None | | |
| GB-A- 1170587 | 12/11/69 | NL-A- | 6616126 | 17/05/67 |
| | | NL-A- | 6616125 | 17/05/67 |
| | | NL-A- | 6616124 | 17/05/67 |
| | | GB-A- | 1170586 | 12/11/69 |
| | | GB-A- | 1170434 | 12/11/69 |
| | | DE-A- | 1524127 | 08/01/70 |
| | | CH-A- | 483061 | 15/12/69 |
| | | US-A- | 3487373 | 30/12/69 |
| | | CH-A- | 495584 | 31/08/70 |
| | | DE-A- | 1524126 | 25/06/70 |
| | | DE-A- | 1524125 | 25/06/70 |
| | | FR-A- | 1513352 | |

For more details about this annex :
see Official Journal of the European Patent Office, No. 12/82